Efficient Detection of Communication-related Performance Anti-patterns in Microservices

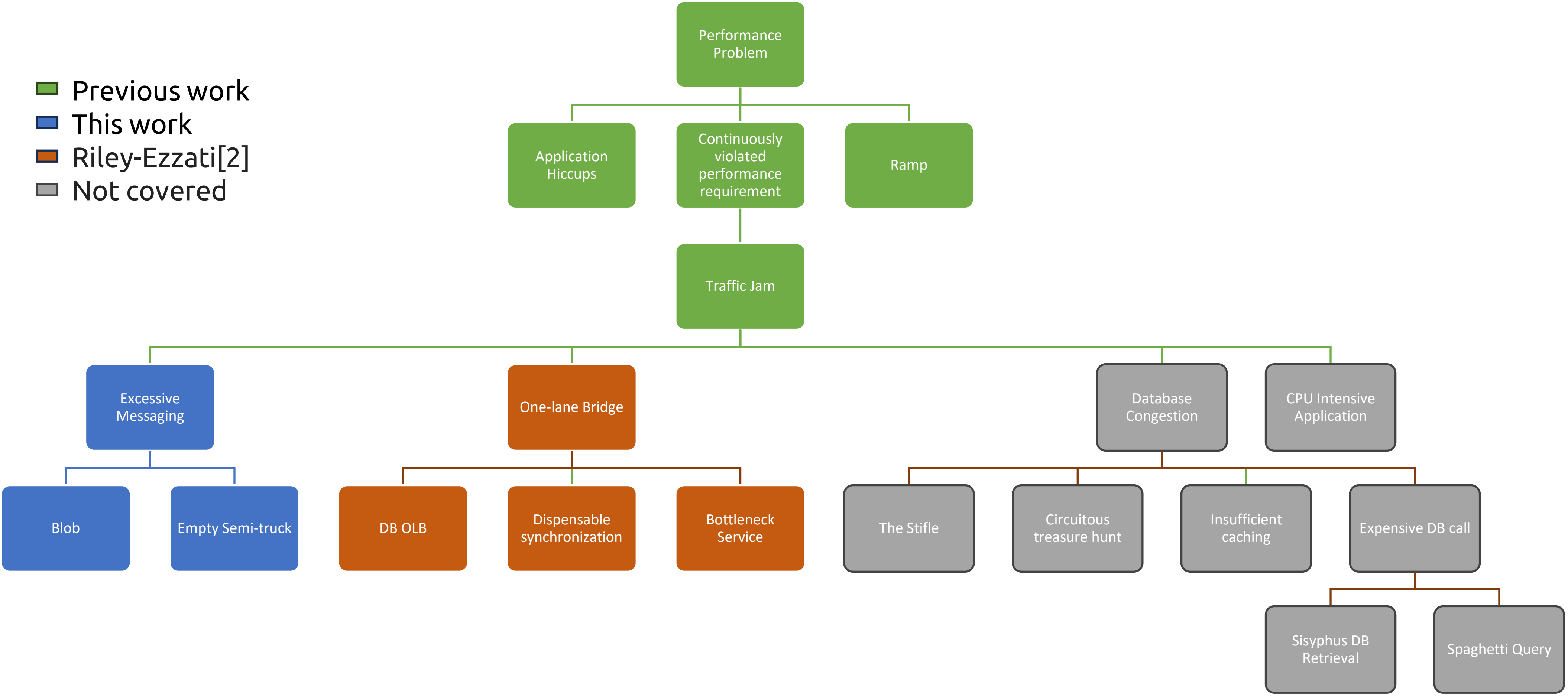
Masoumeh Nourollahi

Polytechnique Montréal

DORSAL Laboratory

Introduction

- Performance anti-patterns (SPAs)
  - Bad practices which result in performance degradation
  - May not cause system failure, but impact performance



🟩 Previous work
🟦 This work
🟧 Riley-Ezzati[2]
⬜ Not covered

Performance Problem
Application Hiccups
Continuously violated performance requirement
Ramp
Traffic Jam
Excessive Messaging
One-lane Bridge
Database Congestion
CPU Intensive Application
Blob
Empty Semi-truck
DB OLB
Dispensable synchronization
Bottleneck Service
The Stifle
Circuitous treasure hunt
Insufficient caching
Expensive DB call
Sisyphus DB Retrieval
Spaghetti Query

Agenda

- Problem statement
- Proposed approach and detection method
- Experiments
- Results
- Discussion and future work

Problem Statement

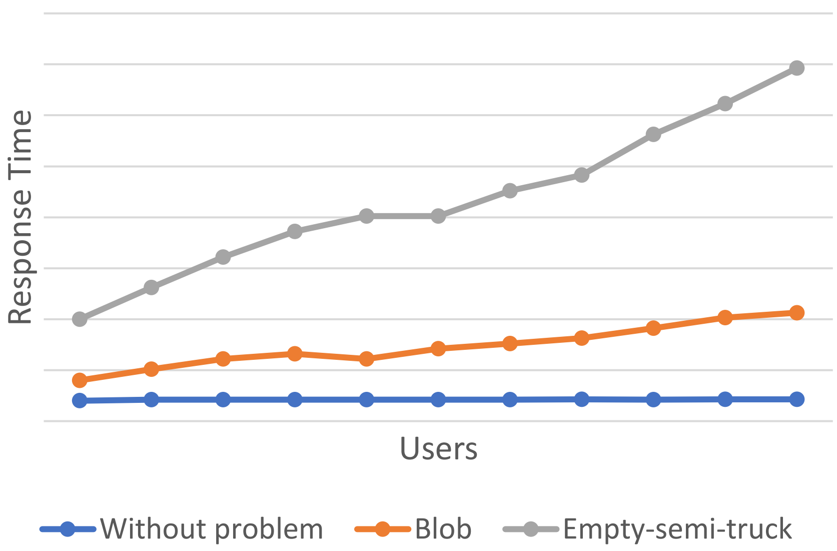
Detecting Inter-Communication Performance Anti-Patterns in Microservice Environments

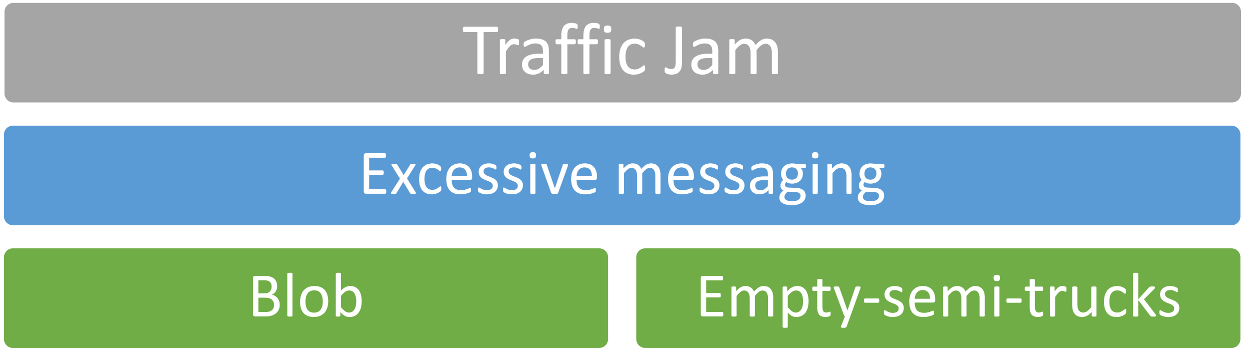
- **Non-intrusive method** for detecting anti-patterns in microservice-based applications.
- **Low overhead**: Minimal cost from trace data collection.
- **Improved accuracy** compared to existing detection methods.
- **Automated pipeline**, easily integrable with CI/CD processes.

Problem statement

- Blob
  - High communication overhead through remote communication between central blob component and other components
- Empty Semi-truck
  - Large number of small messages transmitted between two components as part of a single user request

Response time changes for inter-communication SPAs





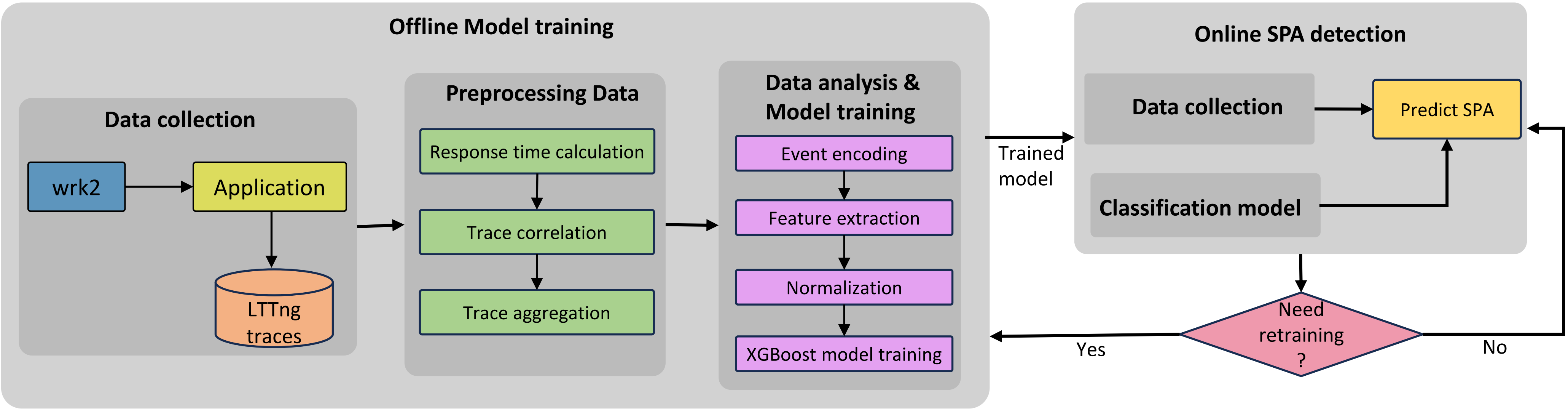
Traffic Jam

Excessive messaging

Blob

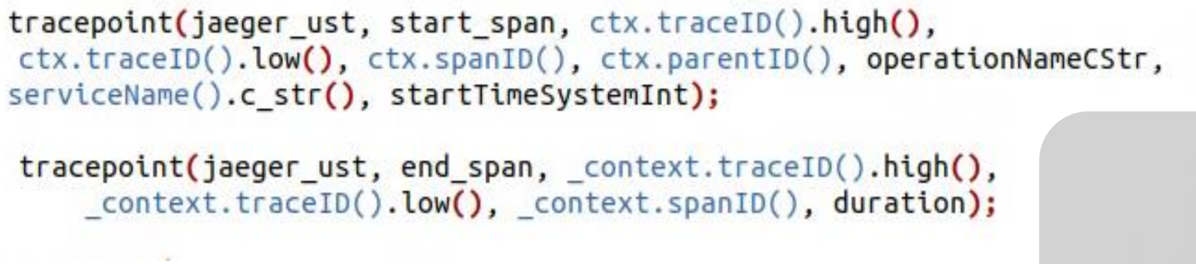
Empty-semi-trucks

Proposed Approach



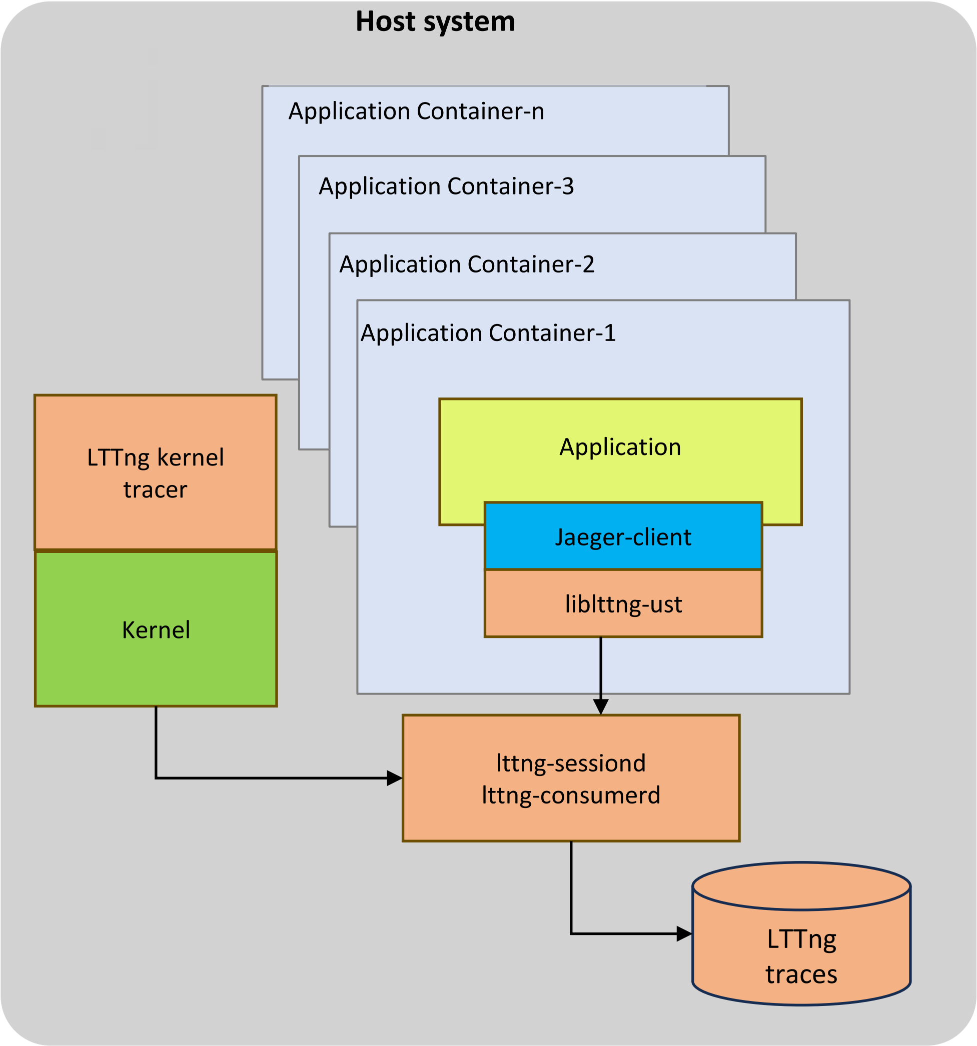
Proposed Approach- Hybrid Tracing

UST tracepoints on jaeger:



```
tracepoint(jaeger_ust, start_span, ctx.traceID().high(),
 ctx.traceID().low(), ctx.spanID(), ctx.parentID(), operationNameCStr,
serviceName().c_str(), startTimeSystemInt);

 tracepoint(jaeger_ust, end_span, _context.traceID().high(),
     _context.traceID().low(), _context.spanID(), duration);
```



Sample user-space and System call traces

Trace collection script:

```
lttng enable-event  --userspace 'jaeger*' --channel=ust-channel --session test-session
lttng enable-event  -k --channel=kernel-channel –syscall recvfrom, recvmsg, recvmmsg, sendto,
sendmsg, sendmmsg
```

OpenTracing Trace sample:

```
Timestamp      Channel   CPU  Event type      Contents  TID   Prio  PID   Source      Binary Location
        Function Location    Source
01:24:47.480 957 960      ust-channel_6_0 6  jaeger_ust:start_span      trace_id_high=0,
trace_id_low=5485338799412484790, span_id=2723353438606276410,
parent_span_id=2205893957549611025, op_name="UploadUserWithUserId",
service_name="user-service", start_time=1726464287480949106, context.packet_seq_num=0,
context.cpu_id=6, context._vtid=111, context._vpid=1 111          1
```

System-call Trace sample:

```
Timestamp      Channel   CPU  Event type      Contents  TID   Prio  PID   Source      Binary Location
        Function Location    Source 21:44:27.237 576 630 kernel-channel_7_0       7
        syscall_exit_recvfrom      ret=12, ubuf=140192631653072, addr=0, addr_len=0,
context.packet_seq_num=0, context.cpu_id=7, context._tid=3487992, context._vtid=52,
context._pid=6076, context._vpid=1      3487992          6076        [net/socket.c:0]
```

Preprocessing- Aggregated trace

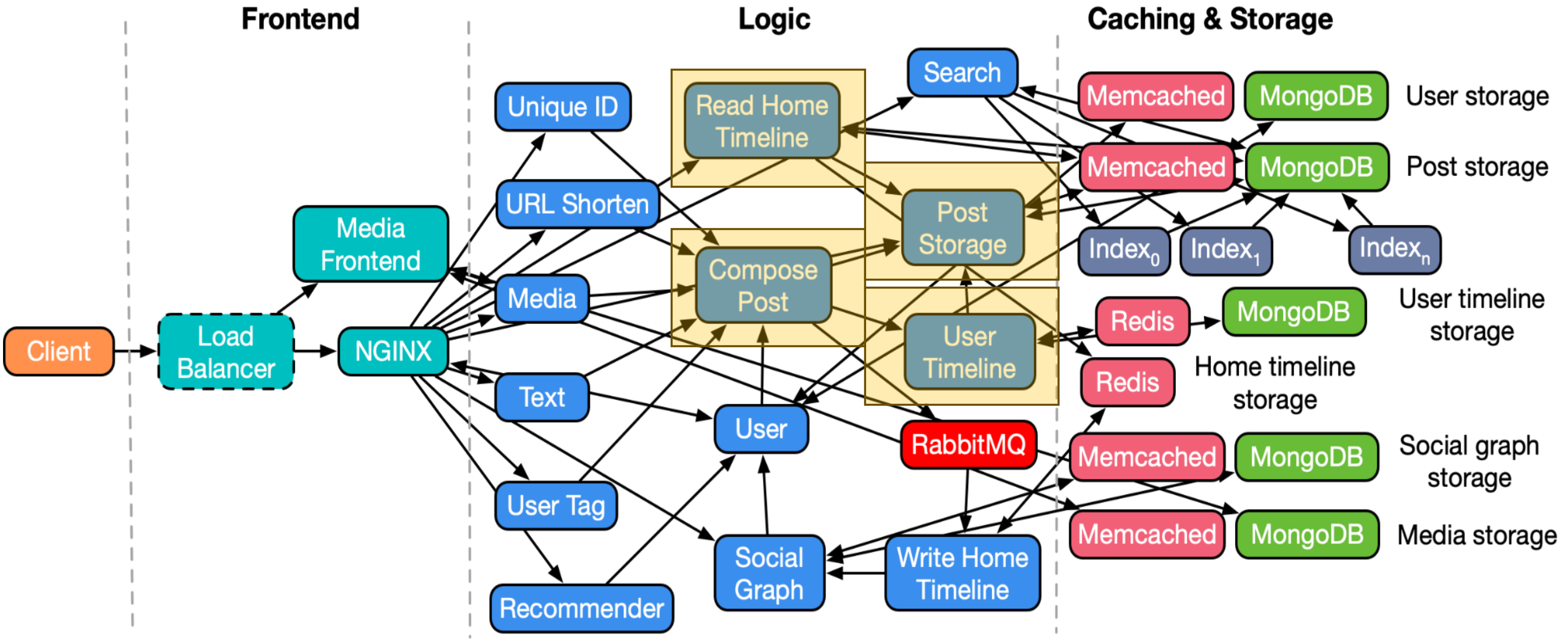
```
#.  name,cur_ts,ret,response_time,span_id,parent_span_id,op_name,service_name,trace_id
1.  jaeger_ust:start_span,1726464134589066276,0,1709015,4706249999899046019,17642707676433399526,ReadUserTimeline,user-timeline-service,9616759882794086189
2.  jaeger_ust:start_span,1726464134589081297,0,24775,17102146431855677904,4706249999899046019,RedisFind,user-timeline-service,9616759882794086189
3.  jaeger_ust:start_span,1726464134589469928,0,16601,1624531274915946197,4706249999899046019,MongoFindUserTimeline,user-timeline-service,9616759882794086189
4.  syscall_entry_sendmsg,1726464134589559979,195,54444,Null,Null,syscall_entry_sendmsg,user-timeline-service,9616759882794086189
5.  syscall_entry_recvfrom,1726464134589633523,0,15908,Null,Null,syscall_entry_recvfrom,user-timeline-service,9616759882794086189
6.  syscall_entry_recvfrom,1726464134590166434,116,19280,Null,Null,syscall_entry_recvfrom,user-timeline-service,9616759882794086189
7.  syscall_entry_recvfrom,1726464134590390202,4,33007,Null,Null,syscall_entry_recvfrom,user-timeline-service,9616759882794086189
8.  syscall_entry_recvfrom,1726464134590456198,123,27597,Null,Null,syscall_entry_recvfrom,user-timeline-service,9616759882794086189
9.  jaeger_ust:start_span,1726464134590526296,0,19078,14929287647722053771,4706249999899046019,ReadPosts,post-storage-service,9616759882794086189
```

Data Analysis and Model Training

- Event encoding and padding
- Feature Extraction
  - Total Response Time, Maximum Response Time, Mean Response Time
  - Total Operations
  - Total Return Values, Maximum Return Value, Mean Return Value
  - Total Service Calls
  - Unique Operation Counts
- Feature Normalization
- Model Training with XGBoost
- Model validity checks
  - Supervised learning (classifier uses labeled trace data)
  - Feature information gain analysis
  - K-fold cross validation

Experiments

- TestBed:
  - DeathStarBench- Social Network application



SPA Injection

Performance anti-pattern	Impacted services	Details
Blob-1	UserTimeline, PostStorage	Moved whole logic of writeUserTimeline operation to PostStorage service
Blob-2	UserTimeline, PostStorage	Moved whole logic of readUserTimeline and writeUserTimeline operation to PostStorage service
Blob-3	UserTimeline, PostStorage, HomeTimeline	Moved whole logic of readUserTimeline, writeUserTimeline, and readHomeTimeline operations to PostStorage service
Empty-semi-trucks-1	ComposePost	In ComposeAndUpload commit each call separately to simulate overhead
Empty-semi-trucks-2	UserTimeline	In readUserTimeline instead of fetching all posts at once, the function uses a for loop where each post is queried from MongoDB individually

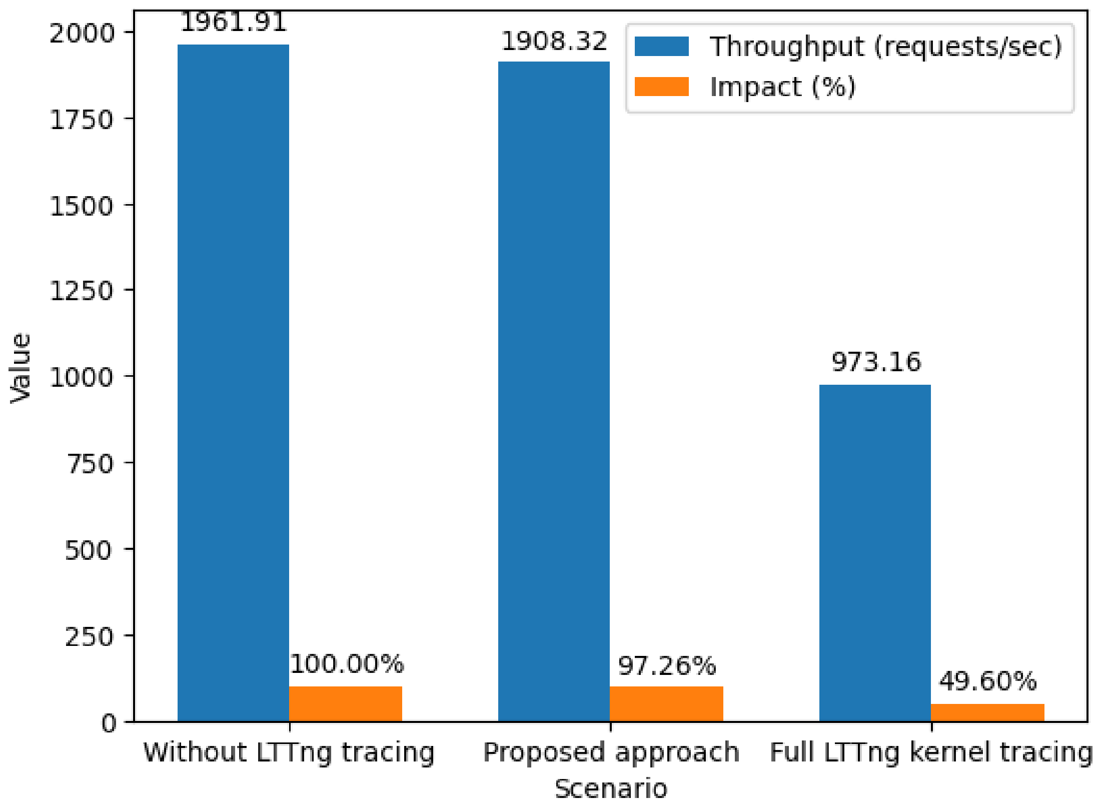
Experiments and Results

#	Scenario	Details
1	Baseline	unchanged application
2	Baseline with system faults	unchanged application with noise
3	Blob-1	Blob-1 injected
4	Blob-2	Blob-2 injected
5	Blob-3	Blob-3 injected
6	Empty-semi-trucks-1	Empty-semi-trucks-1 injected
7	Empty-semi-trucks-2	Empty-semi-trucks-2 injected

Comparison of using different classifiers to detect Blob and Empty-semi-trucks SPAs

Classifier	Precision	Recall	Accuracy
SVM	72.80	72.87	99.81
XGBoost	73.31	73.26	99.98
Random forest	73.31	73.25	99.98

Overhead analysis



Contributions

1. Detecting SPAs by using a non-intrusive, low-cost tracing method.
2. Detection based on selective syscall traces, specifically related to the intended SPA.
3. Benefiting from existing open-tracing instrumentation in the microservice applications.
4. Low-cost SPA detection approach
5. Use of distributed traces jointly with system calls to provide a better characterization of the normal behavior of the system vs anti-pattern effect.
6. Facilitating root-cause analysis
7. Presenting a modern classification method for SPAs, with high detection accuracy (99%)
8. Considering calls between micro-services in the detection model training
9. Advanced baseline calculation method by calculating baseline for each trace type

Future directions

- Using methods like topic modeling to decrease preprocessing efforts
- Add more experiments to build a dataset of performance anti-patterns

References

1. Trubiani, Catia, et al. "Automated detection of software performance antipatterns in Java-based applications." *IEEE Transactions on Software Engineering* (2023).
2. VanDonge, Riley, and Naser Ezzati-Jivan. "N-Lane Bridge Performance Antipattern Analysis Using System-Level Execution Tracing." *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2022.
3. Chalawadi, Ram Kishan. "Automating the characterization and detection of software performance antipatterns using a data-driven approach." (2021).
4. Stefanakos, Ioannis. *Software analysis and refactoring using probabilistic modelling and performance antipatterns*. Diss. University of York, 2021.
5. Smith, Connie U. "Software performance antipatterns in cyber-physical systems." *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. 2020.
6. Avritzer, Alberto, et al. "Scalability assessment of microservice architecture deployment configurations: A domain-based approach leveraging operational profiles and load tests." *Journal of Systems and Software* 165 (2020): 110564.
7. Trubiani, Catia, et al. "Exploiting load testing and profiling for performance antipattern detection." *Information and Software Technology* 95 (2018): 329-345.
8. Heger, Christoph, et al. "Expert-guided automatic diagnosis of performance problems in enterprise applications." *2016 12th European Dependable Computing Conference (EDCC)*. IEEE, 2016.
9. Keck, Philipp, et al. "Antipattern-based problem injection for assessing performance and reliability evaluation techniques." *2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2016.
10. Wert, Alexander, et al. "Automatic detection of performance anti-patterns in inter-component communications." *Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures*. 2014.

Thanks for your attention

Masoumeh.nourollahi@polymtl.ca
https://github.com/mnourollahi